AD-A104 739 CARNEGIE-MELLON UNIV PITTSBURGH PA DEPT OF COMPUTER --ETC F/G 9/2
I/O COMPLEXITY: THE RED-BLUE PEBBLE GAME.(U)
MAR 81 J HONG, H T KUNG N00014-76-C-0370
UNCLASSIFIED CMU-CS-81-111 NL

1 OF 1
AD
A104739

END
DATE
FILMED
10-81
DTIC

LEVEL #

CMU-CS-81-111

ADA104739

I/O COMPLEXITY:
THE RED-BLUE PEBBLE GAME

Hong, Jia-Wei and H. T. Kung

Department of Computer Science
Carnegie-Mellon University
Pittsburgh, Pennsylvania 15213

February 1981
[Last revised March 1981]

DEPARTMENT
of
COMPUTER SCIENCE

Carnegie-Mellon University

CMU-CS-81-111

I/O COMPLEXITY: THE RED-BLUE PEBBLE GAME

Hong, Jia-Wei and H. T. Kung

Department of Computer Science
Carnegie-Mellon University
Pittsburgh, Pennsylvania 15213

February 1981
[Last revised March 1981]

(A preliminary version of this paper is to appear in the *Proceedings of the 13th Annual ACM Symposium on Theory of Computing*, May 1981.)

The research was supported in part by the Office of Naval Research under Contracts N00014-76-C-0370 and N00014-80-C-0236, in part by the National Science Foundation under Grant MCS 78-236-76, and in part by the Defense Advanced Research Projects Agency under Contract F33615-78-C-1551 (monitored by the Air Force Office of Scientific Research). About the authors: Hong, Jia-wei is on leave from Peking Municipal Computing Center, Peking, China and is currently visiting the University of Rochester for the Spring of 1981. H.T. Kung is currently on leave from Carnegie-Mellon University at ESL's Advanced Processor Technology Group in San Jose, California. (ESL is a subsidiary of TRW.) Most of the research for this paper was carried out during the Fall of 1980 when both authors were at CMU.

ABSTRACT

In this paper, the *red-blue pebble game* is proposed to model the input-output complexity of algorithms. Using the pebble game formulation, a number of lower bound results for the I/O requirement are proven. For example, it is shown that to perform the n-point FFT (or the ordinary nxn matrix multiplication algorithm) with a device of O(S) memory, at least $\Omega(n \log n/\log S)$ (or $\Omega(n^3/\sqrt{S})$, respectively) time is needed for the I/O. Similar results are obtained for algorithms for several other problems. All of the lower bounds presented are the best possible in the sense that they are achievable by certain decomposition schemes.

The results in this paper provide insight into the difficult task of balancing I/O and computation in special-purpose system design. For example, for the n-point FFT, the I/O lower bound implies that an S-point device achieving a speed-up ratio O(log S) over the conventional O(n log n) implementation is all that one can hope for.

A

1. Introduction

When a large computation is performed on a small device or memory, the computation must be decomposed into subcomputations. Executing subcomputations one at a time may require a substantial amount of I/O to store or retrieve intermediate results. Very often it is the I/O that dominates the speed of a computation. In fact, I/O is a typical bottleneck for performance at all levels of a computer system. However, to the authors' knowledge the I/O problem was not previously modelled or studied in any systematic or abstract manner. Similar problems were studied only in a few isolated instances [2, 5]. This paper proposes a pebble game, called the *red-blue pebble game*, to model the problem, and presents a number of lower bound results for the I/O requirement. All the lower bounds presented can be shown to be the best possible, in the sense that they are achieved by certain decomposition schemes. The paper is organized according to the techniques used to derive these lower bounds.

In Section 2 we formally define the pebble game and point out its relation to the I/O problem. In Section 3 we show that lower bounds for I/O in the pebble game can be established by studying the so-called *S-partitioning problem*. This is the key result of the paper in the sense that it provides the basis for the derivation of all the lower bounds. In Section 4 we prove a lower bound for the FFT algorithm. Lower bounds in Section 5 are based on the *information speed function*, which measures how fast the number of vertices on which a given vertex "depends" can grow in a directed acyclic graph of a certain type. We demonstrate the dramatic difference between the I/O requirement for the odd-even transposition sorting network and that for the "snake-like" mesh graph. In contrast to the focus of Section 5, Section 6 studies *independent computations* for which there are very little information exchanges among vertices. There we obtain, for example, a lower bound for the ordinary matrix multiplication algorithm. In Section 7 we prove a general theorem on products of graphs. Using this theorem, one can determine the I/O required by a product of graphs, by examining only the individual graphs. A summary and concluding remarks are provided in Section 8.

Results of this paper impose upper bounds on the maximum possible speed-up obtainable with a special-purpose hardware device when the bandwidth of the memory that supplies data to the device remains constant. For example, our lower bound on the I/O requirement for the n-point FFT (Corollary 4.1) implies that an S-point device can achieve a speed-up ratio of at most $O(\log S)$ over the conventional $O(n \log n)$ software implementation. Similarly, for matrix multiplication our result (Corollary 6.2) implies that a $\sqrt{S} \times \sqrt{S}$ device can achieve a speed-up ratio of at most $O(\sqrt{S})$.

2. The Red-Blue Pebble Game and Its Relation to the I/O Problem

As the usual pebble game (see, e.g., [4]), the red-blue pebble game is played on a directed acyclic graph[1]. At any point in the pebble game, some vertices of the graph will have red pebbles, some will have blue pebbles, some will have both red and blue pebbles and the remainder will have no pebbles at all. Following the notation of Pippenger [8], define a *configuration* as a pair of subsets of the vertices, one comprised of just the vertices having red pebbles, and the other just those having blue pebbles. Thus vertices belonging to the intersection of the two sets have both red and blue pebbles on them. The set of *inputs* (or *outputs*) of the graph is some designated set of vertices containing at least those vertices that have no predecessors (or successors, respectively). We assume that the set of inputs is disjoint from that of outputs. For all the examples discussed in the paper, only vertices that have no predecessors (or successors) are assumed to be inputs (or outputs, respectively), except in Section 7 where products of graphs are considered. The *initial* (or *terminal*) configuration is one in which only inputs (or outputs, respectively) have pebbles, and they are all blue pebbles. The rules of the red-blue pebble game are as follows.

R1. (Input) A red pebble may be placed on any vertex that has a blue pebble.

R2. (Output) A blue pebble may be placed on any vertex that has a red pebble.

R3. (Compute) If all the immediate predecessors of a vertex have red pebbles, a red pebble may be placed on that vertex.

R4. (Delete) A pebble (red or blue) may be removed from any vertex.

A *transition* is an ordered pair of configurations, the second of which follows from the first according to one of the rules. A *calculation* is a sequence of configurations, each successive pair of which form a transition. A *complete* calculation is one that begins with the initial configuration and ends with the terminal configuration.

A graph on which the red-blue pebble game is played can model a computation performed on a two-level memory structure, consisting of say, a *fast* memory and a *slow* memory. Vertices represent operations and their results. An edge from one vertex to another indicates that the result of one operation is an operand of the other. An operation can be performed only if all the operands reside in the fast memory. Placing a red pebble using rule R3 corresponds to performing an operation and storing the result in the fast memory. Placing a blue pebble using rule R2 corresponds to storing a copy of a result (currently in the fast memory) into the slow memory, whereas placing a red pebble using R1 corresponds to retrieving a copy of a result (currently in the slow memory) into the fast memory. Removing a red or blue pebble using rule R4 means freeing a memory location in the fast or slow memory, respectively. The maximum allowable number of red

[1] The red-blue pebble game discussed in this paper is not related in any way to the black-and-white pebble game introduced by Cook and Sethi [1].

or blue pebbles on the graph at any point in the game corresponds to the number of words available for use in the fast or slow memory, respectively.

For the purpose of this paper, we assume that the fast memory can hold only S words, where S is a constant, while the slow memory is arbitrarily large. Thus when the pebble game is played on a graph, at most S red pebbles, and any number of blue pebbles, can be on the graph at any time. For any given graph, we are interested in the minimum *I/O time* Q, which is defined by

> Q = the minimum number of transitions according to rule R1 or R2 required by any complete calculation.

For the FFT graph, it is not difficult to prove the following upper bound on Q by the decomposition scheme illustrated in Figure 2-1[2].

Theorem 2.1. For the n-point FFT graph,

$Q \cdot \log S = O(n \log n)$.

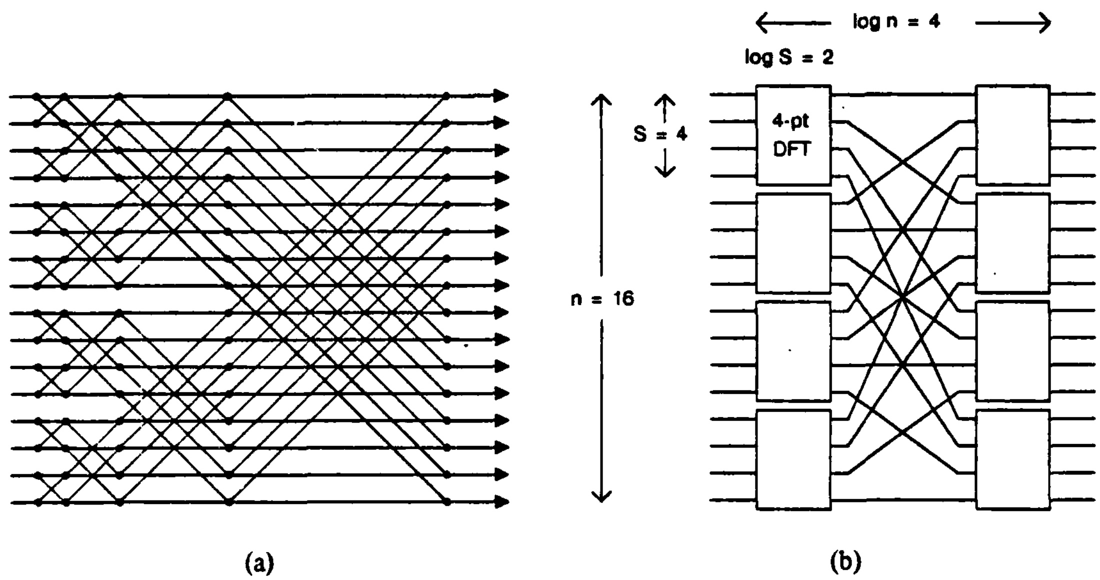

Figure 2-1: (a) the 16-point FFT graph, and (b) decomposing the FFT graph, with n = 16 and S = 4.

However, for proving tight lower bounds on Q, we found that it was difficult to work with the red-blue pebble game directly. Instead we study the *S-partitioning problem*, which is a "static" problem in the sense

[2] All logarithms used in this paper are to base 2.

that it does not apply rules on-the-fly as in a game. We show that lower bounds for the S-partitioning problem can be translated into lower bounds on Q for the red-blue pebble game.

3. The S-Partitioning Problem and the Key Lemma

In this section we show that every complete calculation of the red-blue pebble game on a directed acyclic graph defines a partition of the graph. Let $G = (V, E)$ be a directed acyclic graph where V and E are the vertex and edge sets of G, respectively. A family of subsets of V, $\{V_1, V_2, \ldots, V_h\}$, is called an *S-partition* of G for some positive integer S if the following four properties hold.

P1. The V_i's are disjoint and $\cup_{i=1}^{h} V_i = V$.

P2. For each V_i, $1 \le i \le h$, there exists a *dominator set* D_i for V_i that contains at most S vertices. (A dominator set for V_i is defined to be a set of vertices in V such that every path from an input of G to a vertex in V_i contains some vertex in the set.)

P3. For each V_i, $1 \le i \le h$, the *minimum set* M_i of V_i has at most S vertices. (The minimum set of V_i is defined to be the set of those vertices in V_i that do not have any sons belonging to V_i.)

P4. There is no cyclic dependence among vertex sets in $\{V_1, V_2, \ldots, V_h\}$. (A vertex set V_i is said to *depend* on another vertex set V_j if there is an edge in E from a vertex in V_j to a vertex in V_i.)

Theorem 3.1. Let $G = (V, E)$ be a directed acyclic graph. Any complete calculation of the red-blue pebble game on G, using at most S red pebbles, is associated with a 2S-partition of G such that

$$S \cdot h \ge q \ge S \cdot (h - 1),$$

where q is the I/O time required by the complete calculation, and h is the number of vertex sets in the 2S-partition.

Proof: Denote by C any complete calculation. We can divide C into a sequence of h consecutive *subcalculations*, $C_1, C_2, \ldots, C_h$, for some h such that in each C_i, $1 \le i \le h-1$, there are exactly S transitions using rule R1 or R2, and in C_h there are no more than S such transitions. For $i = 1, \ldots, h$, define V_i to be the largest vertex set in which each vertex satisfies the following three properties.

(i) During subcalculation C_i it has a red pebble placed on it using rule R1 or R3.

(ii) At the end of subcalculation C_i, it either has red pebbles, or blue pebbles that are placed on it during C_i, or has a son in V_i.

(iii) It does not belong to any V_j with $j < i$.

We claim that the family $\{V_1, V_2, \ldots, V_h\}$ is a 2S-partition of G. First we show that property P1 holds. By (iii) it follows immediately that the V_i's are disjoint. In the following we show that every vertex in V belongs to some V_i. Suppose that a vertex, which is not an input, has a red or blue pebble on it at the end of some subcalculation C_i. Then there must exist a subcalculation C_j, $j \le i$, during which the vertex has a red pebble placed on it using rule R3, and at the end of C_j it either remains to have the red pebble or has a blue pebble that is placed on it during C_j. This implies

that the vertex belongs to V_k for some $k \leq j$. Similarly one can show that if an input has a red pebble on it at the end of C_i, then it must belong to V_k for some $k \leq i$. Because calculation C is a complete calculation, all outputs have blue pebbles on them at the end of the last subcalculation C_h; thus they all belong to $\cup_{i=1}^{h} V_i$. Consider now any immediate predecessor *u* of an output *v*. Suppose that *v* belongs to V_i. Then *v* cannot have any pebble on it at the beginning of C_i and thus must have a red pebble placed on it using R3 during C_i. This implies that we have one of the following two cases:

> Case 1: Vertex *u* has a red pebble on it at the end of subcalculation C_{i-1}. Then by reasons stated above, *u* belongs to some V_j, $j \leq i-1$.
>
> Case 2: Vertex *u* has a red pebble placed on it using rule R1 or R3 during C_i. If *u* does not belong to any V_j with $j < i$, then because *u* has a son *v* in V_i, *u* itself must belong to V_i.

We have shown that all the immediate predecessors of outputs belong to $\cup_{i=1}^{h} V_i$. Similarly, we can show that all the immediate predecessors of the immediate predecessors of outputs belong to $\cup_{i=1}^{h} V_i$. Property P1 follows by induction. Note that both Case 1 and Case 2 above imply that if V_i depends on V_j then $j < i$. Therefore there cannot be any cyclic dependence among V_i's, and thus property P4 holds. For proving property P2 for any V_i, $1 \leq i \leq h$, we consider two subsets of V, V_R and V_{BR}, which are defined as follows.

- V_R consists of those vertices that have red pebbles placed on them just before subcalculation C_i begins.
- V_{BR} consists of those vertices that have blue pebbles placed on them just before subcalculation C_i begins and have red pebbles placed on them according to rule R1 during C_i.

It is easy to see that by property (i) in the definition of V_i, $V_R \cup V_{BR}$ forms a dominator set for V_i. Since there can be at most S red pebbles on G at any time, we have

$$|V_R| \leq S.$$

The fact that at most S transitions can use rule R1 during C_i implies that

$$|V_{BR}| \leq S.$$

Thus

$$|V_R \cup V_{BR}| \leq |V_R| + |V_{BR}| \leq 2S.$$

We have shown that $\{V_1, V_2, \ldots, V_h\}$ satisfies property P2. The proof of property P3 is similar. By property (ii) in the definition of V_i, we know that at the end of subcalculation C_i, every vertex in M_i, the minimum set of V_i, has red pebbles, or blue pebbles that are placed on it during C_i. Since there can be at most S vertices having red pebbles placed on them at any time, and at most S vertices having blue pebbles placed on them according to rule R2 during C_i, the minimum set M_i can have at most 2S vertices. We have shown that $\{V_1, V_2, \ldots, V_h\}$ is a 2S-partition of G. The theorem follows by noting that corresponding to each V_i, $1 \leq i \leq h-1$, exactly S transitions using R1 or R2 are performed and to V_h, no more than S such transitions are performed. □

Let

P(S) = the minimum number of vertex sets that any S-partition of G must have.

We have, by Theorem 3.1, the key lemma of the paper.

Lemma 3.1. For any directed acyclic graph G, the minimum I/O time satisfies

$$Q \geq S \cdot (P(2S) - 1).$$

Using this lemma, lower bounds for P can be translated immediately into lower bounds for Q.

4. Lower Bounds for the FFT Computation

In this section we establish a lower bound on the minimum I/O time Q for the n-point FFT graph (see Figure 2-1(a)), by proving a lower bound on P.

Define an *S-dominator partition* of a graph $G = (V, E)$ to be a family of subsets of V, $\{V_1, V_2, \ldots, V_h\}$, satisfying properties P1, P2 and P4 of an S-partition, but not necessarily property P3. Let

$P_D(S)$ = the minimum number of vertex sets that any S-dominator partition of G must have.

Then clearly $P_D(S) \leq P(S)$, since any S-partition is also an S-dominator partition. The following theorem establishes a lower bound on $P_D(S)$, and thus a lower bound on P(S).

Theorem 4.1. Suppose that $S \geq 2$. The minimum number of vertex sets that any S-dominator partition of the n-point FFT graph must have satisfies

$$P_D(S) = \Omega((n \log n)/(S \log S)).$$

Proof: Since there are a total of $\Theta(n \log n)$ vertices in the n-point FFT graph, it suffices to prove that any vertex set U that has a dominator set of size no more than S, $S \leq n$, can have at most $S \log S + S$ vertices. We shall show this by induction on n. The assertion holds trivially for the case when $n = 2$. Assume now that it holds for the m-point FFT for any $m < n$. We want to show that it holds for $m = n$. Consider the n-point FFT graph. We partition its vertex set into four disjoint sets A, B, C and D such that sets C and D equally partition the set of outputs, and sets A and B equally separate the rest of the vertices. See Figure 4-1 below. Let d_A, d_B, d_C or d_D be the number of elements in the dominator set that belong to sets A, B, C or D respectively. Let u_A, u_B, u_C or u_D be the number of elements in the vertex set U that belong to sets A, B, C or D, respectively. It is easy to see that elements in set A that are also in the dominator set form a dominator set for set $U \cap A$. Thus by the induction hypothesis,

$$u_A \leq d_A \log d_A + d_A. \tag{1}$$

Similarly, we have

$$u_B \leq d_B \log d_B + d_B. \tag{2}$$

Let R_A or R_B be the set of those horizontal paths from inputs to outputs on which there are vertices in the dominator set that belong to A or B, respectively. Then

$$|R_A| \leq d_A \text{ and } |R_B| \leq d_B. \tag{3}$$

For each vertex in $U \cap C$ or $U \cap D$, it either belongs to the dominator set or one of its immediate predecessors is on a horizontal path belonging to R_A and the other on one belonging to R_B. Therefore

$$u_C \leq d_C + \min(|R_A|, |R_B|) \text{ and } u_D \leq d_D + \min(|R_A|, |R_B|),$$

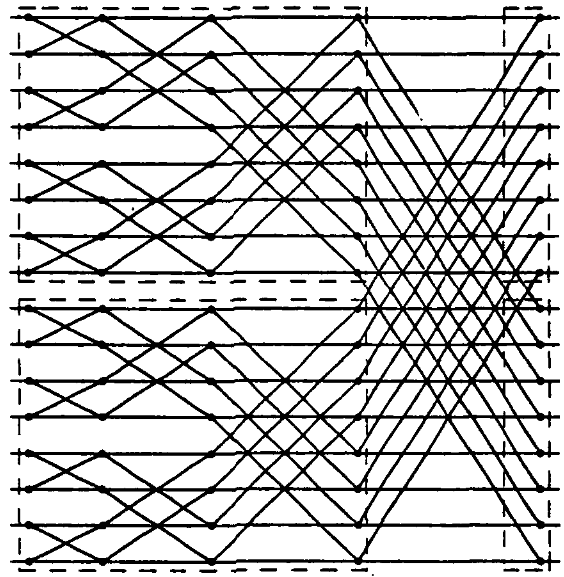

Figure 4-1: Partitioning the FFT graph for the induction proof.

from which we have

$$u_C + u_D \leq d_C + d_D + 2\min(|R_A|, |R_B|). \tag{4}$$

By (1), (2), (3) and (4),

$$u_A + u_B + u_C + u_D \leq [d_A \log d_A + d_B \log d_B + 2\min(d_A, d_B)] + d_A + d_B + d_C + d_D.$$

Since $d_A + d_B \leq S - d_C - d_D$, we have

$$u_A + u_B + u_C + u_D \leq (S-d_C-d_D) \log (S-d_C-d_D) + S \leq S \log S+S,$$

which completes the induction proof. □

By Lemma 3.1 we have the following lower bound result.

Corollary 4.1. For the n-point FFT graph,

$$Q \cdot \log S = \Omega(n \log n).$$

Thus the I/O time for the n-point FFT when executed on a special-purpose device with S words of memory is at least $\Omega(n \log n/\log S)$, implying that the maximum-possible speed-up ratio over the usual $O(n \log n)$ implementation is at most $O(\log S)$. This upper bound on the speed-up ratio holds no matter how fast the the device may be, since it is a consequence of the I/O consideration. The upper bound can be reduced only if the bandwidth of the memory that supplies data to the special-purpose device is increased. A systolic device that distributes S words of memory in a linear processor array and achieves $\Theta(\log S)$ speed-up for the FFT is described by Kung [7].

5. Lower Bounds Based on Information Speed Functions

Many "regular" graphs $G = (V, E)$ have the property that all inputs can reach all outputs through vertex-disjoint paths. In the proof of Theorem 4.1 we have already noted that the FFT graph has this property. In the current section, this type of graph will be considered. The vertex-disjoint paths from inputs to outputs will be called *lines*, for simplicity. We say that the *information speed function* is $\Omega(F(d))$ if for any two vertices *u*, *v* on the same line that are d apart, there are at least $F(d)$ vertices in the graph satisfying the following two properties.

F1. At most one of these vertices can belong to a single line.

F2. Each of these vertices belongs to a path connecting *u* and *v*.

The following theorem shows that lower bounds on Q can be obtained from lower bounds on F or upper bounds on F^{-1}.

Theorem 5.1. For any graph where all inputs can reach all outputs through vertex-disjoint paths, if the information speed function is $\Omega(F(d))$ where F is monotonically increasing and F^{-1} exists, then

$$Q \cdot F^{-1}(S) = \Omega(L),$$

where L is the total number of vertices on the vertex-disjoint paths or the lines.

Proof: As in the proof of Theorem 4.1, we will establish

$$P_D(S) = \Omega(L / (S \cdot F^{-1}(S)))$$

by showing that any vertex set U in a S-dominator partition can have at most $O(S \cdot F^{-1}(S))$ vertices on the lines. Note that vertices in U can be on at most S lines, since the lines are vertex-disjoint and U has a dominator set of size at most S. The theorem follows from the claim that on any line there can be at most $F^{-1}(S) + 1$ vertices in U. Suppose that the claim is false for some line. Then on this line there are two vertices *u* and *v* in U that are $F^{-1}(S) + 1$ apart. Consequently, there are $F(F^{-1}(S) + 1)$ vertices satisfying properties F1 and F2. If any of these vertices belongs to another vertex set U' in the S-dominator partition, then by property F2 there will be a cyclic dependence among vertex sets in the S-dominator partition, violating property P4 in Section 3. Therefore all of these $F(F^{-1}(S) + 1)$ vertices, which form a set of more than S vertices, belong to U, and by property F1 they belong to distinct lines. This is a contradiction, since vertices in U can be on at most S lines. □

Corollary 5.1. For the odd-even transposition sorting network (see, e.g., [6]) for sorting n-element runs,

$$Q \cdot S = \Omega(n^2),$$

for any $S < n$.

Proof: Consider the sub-network that includes only half of the inputs and outputs, as shown in Figure 5-1. It is easy to see that we can assume the sub-network has n/2 lines with $L = \Theta(n^2)$ and $F(d) = d/2$ for $d \leq n$. □

Corollary 5.2. For the mxn snake-like directed mesh as shown in Figure 5-2,

$$Q = \Omega(mn),$$

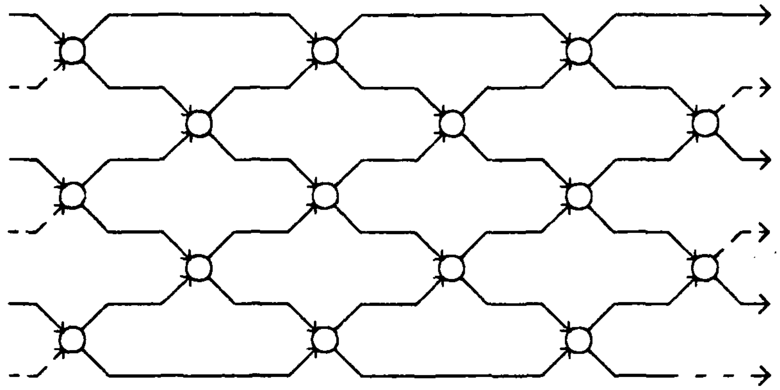

Figure 5-1: The odd-even transposition sorting network, where each "o" is a comparator.

for any $S < m$.

Proof: Consider as lines all the horizontal vertex-disjoint paths from inputs to outputs. It is easy to see that we can assume $F(d) = m$ for any $d \geq 2$. Let U be any vertex set in an S-dominator partition of the graph. As in the proof of Theorem 5.1, we note that vertices in U can be on at most S lines, and that on any line there can be at most two vertices in U. Therefore, U can have at most O(S) vertices, and thus $P_D(S)$ (or $P(S)$) $= \Omega(mn/S)$. The corollary follows from Lemma 3.1. □

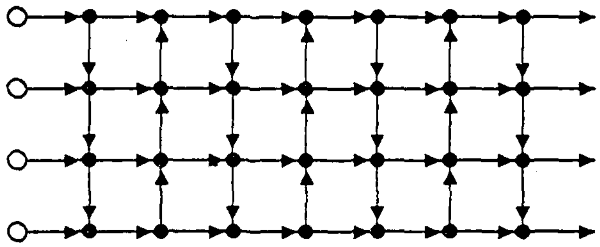

Figure 5-2: The snake-like directed mesh.

Straightforward decomposition schemes will show that lower bounds in the above corollaries are best possible. We note that when S increases the I/O requirement Q for the odd-even transposition sorting network decreases at the rate of 1/S, whereas that for the snake-like directed mesh remains unchanged essentially. We say that graphs like the latter are *indecomposable.*

6. Independent Evaluation of Multivariate Expressions

Given values for indeterminates $x_1, \ldots, x_n$, the problem is to evaluate multivariate polynomial expressions $y_i = y_i(x_1, \ldots, x_n)$, $i = 1, 2, \ldots, m$. Assume that each y_i is a sum of at least two terms and in each y_i, all the terms are distinct and have degrees $\leq D$. An example of such a problem is matrix multiplication, where $D = 2$. An *independent evaluation* of y_i's is an algorithm or a directed acyclic graph with inputs x_i's and outputs y_i's satisfying the following properties.

E1. In the evaluation of each y_i, all (and only) those product terms which appear in the fully distributed expression of y_i are computed first by multiplications, and then using these product terms y_i is formed through a *summation tree* by additions or subtractions only. In particular, no multiplication can be performed after an addition or subtraction.

E2. Internal vertex sets of the summation trees for all the y_i's are disjoint from each other, that is, none of the internal vertices in one tree appears as an internal vertex in another. (Thus, evaluations of y_i's are *independent* from each other.)

Let X be any set of x_i's or products in x_i's. For any output y_i, define $h(y_i, X)$ as the number of terms in y_i that can be obtained from X directly or by multiplying elements in X. For any $Y \subseteq \{y_1, \ldots, y_m\}$ we further define

$$h(Y, X) = \textstyle\sum_{y \in Y} h(y, X).$$

For example, if $y_1 = x_1x_2 + x_3^2x_1$, $y_2 = x_1^2x_2^2 + x_1x_3^4$, $Y = \{y_1, y_2\}$, and $X = \{x_1, x_2^2, x_3^2\}$, then $h(y_1, X) = 1$, $h(y_2, X) = 2$, and $h(Y, X) = 3$. Define the *S-combination number* to be

$$H(S) = \max\{h(Y, X) \mid |Y| \leq S, |X| \leq S\}.$$

We have the following result.

Theorem 6.1. Suppose that $H(S) = \Omega(S)$. Then for any independent evaluation of a multivariate expression of degree $\leq D$,

$$Q \cdot D \cdot H(S)/S = \Omega(|V|),$$

where $|V|$ is the total number of vertices in the graph corresponding to the independent evaluation.

Proof: Let $\{V_1, V_2, \ldots, V_h\}$ be an S-partition of the graph associated with the independent evaluation. We shall prove the following.

(i) Each V_i, $1 \leq i \leq h$, can have at most $H(S) + 2S$ *internal vertices.* (An internal vertex is defined to be a vertex belonging to the internal vertex set of some summation tree.)

(ii) There are at least $|V|/(2D)$ internal vertices in the graph.

By property P3 in the definition of S-partition, the minimum set of V_i has at most S vertices. This implies that V_i can have nonempty intersections with internal vertex sets of at most S summation trees, since by E2 each of such intersections has at least one distinct vertex in the minimum set. Thus, to bound the number of internal vertices that V_i can have, we need only consider summation trees for S y_i's. By property P2 of S-partition, we note that V_i has a dominator

set D_i of size no more than S. By the definition of H(S), from D_i one can form at most H(S) terms appearing in the S y_i's. These terms, together with possible vertices in D_i that are already internal vertices, can generate at most H(S) + 2S internal vertices. We have shown (i). To prove (ii), let A be the total number of internal vertices in the graph corresponding to the independent evaluation. Then the total number of external vertices, or terms, in all the summation trees, is no greater than 2A. Each product term requires at most D − 1 multiplications; thus the total number of vertices |V| in the graph satisfies:

$$|V| \leq 2A(D-1) + A \leq 2AD.$$

This proves (ii). It follows from (i) and (ii) that

$$h \geq (|V|/2D) / (H(S) + 2S),$$

and by Lemma 3.1,

$$Q = \Omega(S \cdot |V| / (D \cdot (H(2S)+2S))).$$

The theorem follows from the assumption that $H(S) = \Omega(S)$. □

Corollary 6.1. For the ordinary matrix-vector multiplication algorithm for multiplying an mxn matrix with an n-vector,

$$Q \cdot S = \Omega(mn),$$

assuming that entries in the matrix can be generated on-the-fly and thus are not required to be input.

Proof: The corollary follows immediately by noting that $H(S) = \Theta(S^2)$ and $D = 1$. □

Lemma 6.1. For matrix-matrix multiplication,

$$H(S) = \Theta(S^{3/2}).$$

Proof: We shall only prove $H(S) = O(S^{3/2})$, since it is trivial to see $H(S) = \Omega(S^{3/2})$. Consider the matrix multiplication, AB = C. Let W be any set of entries in A and B, with $|W| \leq S$. Partition A into two classes as follows. Class A_d consists of all rows in A, each of which has at least $\sqrt{S}$ entries in W, and class A'_d consists of the rest of rows in A. Accordingly, matrix C is partitioned into two classes, A_dB and A'_dB. Since A_d can have at most $\sqrt{S}$ rows, and since in any row of A_dB an entry in B can appear at most once (and B has no more than S entries in W), the maximum number of terms in A_dB that can be obtained by multiplying elements in W is at most $S \cdot \sqrt{S} = S^{3/2}$. For entries in A'_dB, each of them can be obtained by multiplying at most $\sqrt{S}$ elements in W, since each row in A'_d has at most $\sqrt{S}$ elements in W. Therefore, in any S entries of A'_dB, there are at most $S \cdot \sqrt{S} = S^{3/2}$ terms that can be obtained by multiplying elements in W. □

By Theorem 6.1 and Lemma 6.1, we have the following result.

Corollary 6.2. For the ordinary matrix-matrix multiplication algorithm for multiplying mxk and kxn matrices,

$$Q \cdot \sqrt{S} = \Omega(mkn).$$

7. Lower Bounds for Products of Graphs

As demonstrated in Sections 4 and 5, one can establish lower bounds on Q by proving upper bounds on the size of any vertex set that has a dominator set of size at most S. This is equivalent to proving lower bounds on

$D(n) =$ the minimum size of a dominator set for any vertex set having no less than n vertices.

In this section we show that lower bounds on $D(n)$ for the product of two graphs can be obtained from lower bounds on $D(n)$ for individual graphs. (See, for example, [3] for the definition of the product of two graphs.) Let G_1xG_2 be the product of G_1 and G_2. A vertex $(v_1, v_2) \in G_1xG_2$ is defined to be an input (or output) of G_1xG_2 if v_1 is an input of G_1 or v_2 is an input of G_2, (or, respectively, v_1 is an output of G_1 and v_2 is an output of G_2.) Of course $D(n)$ depends on the graph on which it defines; we use $D_1(n)$, $D_2(n)$ and $D(n)$ to distinguish the case when the graph is G_1, G_2 and G respectively.

Lemma 7.1. If f is a positive function such that $f(x)/x$ is non-increasing, $\sum a_i \geq T_1T_2$, and $0 \leq a_i \leq T_2$, then

$$\sum f(a_i) \geq T_1 f(T_2).$$

Proof:

$$\sum f(a_i) \geq \sum a_i f(T_2)/T_2 \geq T_1 f(T_2). \qquad \square$$

Theorem 7.1. (*The Production Theorem for Dominators*)
If $D_i(n) = \Omega(d_i(n))$ where d_i, $i = 1, 2$, is a positive, non-decreasing function such that $d_i(x)/x$ is non-increasing, then

$$D(n_1 n_2) = \Omega(\min\{n_1 \cdot d_2(n_2), n_2 \cdot d_1(n_1)\}).$$

Proof: Let W be a subset in V_1xV_2 of size n_1n_2. Define

$U_2 =$ the set of vertices p_2 in V_2 for which $|W \cap (V_1x\{p_2\})| \geq n_1$,

and

$$U_2' = V_2 - U_2.$$

Clearly, we have $|U_2| \leq n_2$ giving

$$|W \cap (\{p_1\}xU_2)| \leq n_2, \tag{5}$$

and for $p \in U_2'$,

$$|W \cap (V_1 x \{p_2\})| < n_1. \tag{6}$$

One of the following two cases must hold.

Case 1. $|W \cap (V_1xU_2)| \geq n_1n_2/2$.

Let p_1 be any vertex in V_1. Any dominator set for $W \cap (\{p_1\}xV_2)$ is of size at least $d_2(|W \cap (\{p_1\}xV_2)|)$. Thus the size of any dominator set for W satisfies:

$$D(n_1n_2) \geq \sum_{p_1 \epsilon V_1} d_2(|W \cap (\{p_1\}xV_2)|).$$

Since U_2 is a subset of V_2 and d_2 is a non-decreasing function, we have

$$D(n_1n_2) \geq \sum_{p_1 \epsilon V_1} d_2(|W \cap (\{p_1\}xU_2)|).$$

By the definition of Case 1,

$$\sum_{p_1 \epsilon V_1} |W \cap (\{p_1\}xU_2)| \geq n_1n_2/2. \tag{7}$$

By Lemma 7.1, it follows from (5) and (7) that

$$\sum_{p_1 \epsilon V_1} d_2(|W \cap (\{p_1\}xU_2)|) \geq n_1 \cdot d_2(n_2)/2,$$

implying

$$D(n_1n_2) \geq n_1 \cdot d_2(n_2)/2.$$

Case 2. $|W \cap (V_1xU_2')| > n_1n_2/2$.

Let p_2 be any vertex in V_2. Any dominator set for $W \cap (V_1x\{p_2\})$ is of size at least $d_1(|W \cap (V_1x\{p_2\})|)$. Thus the size of any dominator set for W satisfies:

$$D(n_1n_2) \geq \sum_{p_2 \epsilon V_2} d_1(|W \cap (V_1x\{p_2\})|.$$

Since U_2' is a subset of V_2, we have

$$D(n_1n_2) \geq \sum_{p_2 \epsilon U_2'} d_1(|W \cap (V_1x\{p_2\})|).$$

By the definition of Case 2,

$$\sum_{p_2 \epsilon U_2'} |W \cap (V_1x\{p_2\})| \geq n_1n_2/2, \tag{8}$$

By Lemma 7.1, it follows from (6) and (8) that

$$\sum_{p_2 \epsilon U_2'} d_1(|W \cap (V_1x\{p_2\})|) \geq n_2 \cdot d_1(n_1)/2,$$

implying

$$D(n_1n_2) \geq n_2 \cdot d_1(n_1)/2. \qquad \square$$

Let $L_1 = \{V, E\}$ be a directed line where $V = \{1, 2, ..., m\}$, and $E = \{(i, i+1) \mid i = 1, 2, ..., m-1\}$, with unique input "1" and output "m." We have $D_{L_1}(n) = 1$ for any $n \leq m$. See Figure 7-1.

Let $L_2 = L_1xL_1$. Then

$$D_{L_2}(n^2) = \Omega(\min\{1 \cdot n, 1 \cdot n\}),$$

giving

$$D_{L_2}(n^2) = \Theta(n).$$

Let $L_3 = L_2xL_1$. Then

$$D_{L_3}(n^3) = \Omega(\min\{n \cdot n, n^2 \cdot 1\}),$$

giving

$$D_{L_3}(n^3) = \Theta(n^2).$$

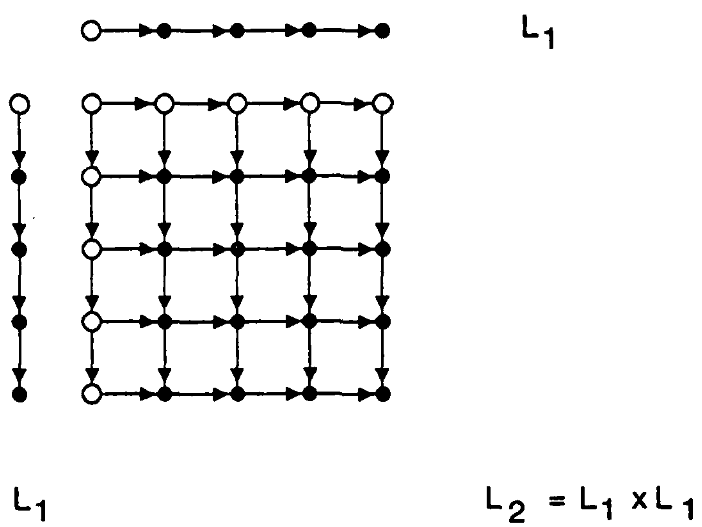

Figure 7-1: The product of two directed lines, where each "o" represents an input.

Let $L_d = L_1 \times \ldots \times L_1$, that is, L_d is the product of d L_1's. Then similarly,

$$D_{L_d}(n^d) = \Theta(n^{d-1}). \tag{9}$$

Corollary 7.1. For the product L_d with $d \geq 2$,

$$Q \cdot S^{1/(d-1)} = \Omega(m^d).$$

Proof: By (9), the maximum size of any vertex set that has a dominator set of size at most S is $O(S^{d/(d-1)})$. Since there are a total of m^d vertices in L_d, we have

$$P(S) = \Omega(m^d/S^{d/(d-1)}),$$

by which the Corollary follows from Lemma 3.1. □

We have a similar product theorem for separators of a graph. For the special case L_d, bounds on the sizes of minimum separators have been established by A. L. Rosenberg [9].

8. Summary and Concluding Remarks

To compare I/O requirements for different algorithms, we propose the use of the following measure. The *decomposability factor* $\lambda(S)$ of an algorithm or graph $G = (V, E)$ is defined to be the ratio between the *sequential time* of the algorithm, that is $|V|$, and the minimum I/O time Q when assuming S red pebbles are used. Thus,

$$Q \cdot \lambda(S) = |V|.$$

For a given algorithm, $|V|$ is fixed. We see that the larger the $\lambda(S)$ is, the less the I/O is required. A summary of results of this paper on specific algorithms or graphs, expressed in terms of bounds on $\lambda(S)$, is as follows:

Algorithms or Graphs	$\lambda(S)$
Matrix-vector multiplication (ordinary algorithm)	$\Theta(S)$
Odd-even transposition sorting network	$\Theta(S)$
Matrix-matrix multiplication (ordinary algorithm)	$\Theta(\sqrt{S})$
L_d, $(d \geq 2)$	$\Theta(S^{1/(d-1)})$
FFT	$\Theta(\log S)$
Snake-like directed mesh	$\Theta(1)$

It is also possible to establish upper bounds on $\lambda(S)$ for a class of algorithms for solving a given problem. For example, it has been shown recently that for *any* sorting algorithm based on the decision tree model, $\lambda(S) = O(\log S)$ [10].

The problem of establishing bounds on $\lambda(S)$ is closely related to several other graph partitioning problems. We intend to work on some of these partitioning problems in the future, and show how they are related to the I/O complexity problem addressed in this paper.

Acknowledgments

The authors would like to thank S. W. Song for his comments on this paper.

References

[1] Cook, S.A. and Sethi, R.
Storage Requirements for Deterministic Polynomial Time Recognizable Languages.
J. Comp. and Sys. Sci. 13:25-37, 1976.

[2] Floyd, R.W.
Permuting Information in Idealized Two-Level Storage.
In Miller, R.E. and Thatcher, J.W. (editor), *Complexity of Computer Computations*, pages 105-109. Plenum Press, New York, 1972.

[3] Harary, F.
Graph Theory.
Addison-Wesley, Reading, Massachusetts, 1969.

[4] Hopcroft, J.E., Paul, W. and Valiant, L.G.
On Time Versus Space.
Journal of the ACM 24:332-337, 1977.

[5] Kernighan, B.W. and Lin, S.
An Effective Heuristic Procedure for Partitioning Graphs.
Bell Systems Technical Journal 49:291-308, February, 1970.

[6] Knuth, D. E.
The Art of Computer Programming. Volume 3: *Sorting and Searching.*
Addison-Wesley, Reading, Massachusetts, 1973.

[7] Kung, H.T.
Special-Purpose Devices for Signal and Image Processing: An Opportunity in VLSI.
In *Proceedings of the SPIE, Vol. 241, Real-Time Signal Processing III*, pages 76-84. The Society of Photo-Optical Instrumentation Engineers, July, 1980.

[8] Pippenger, N.
Pebbling.
In *Proceedings of the Fifth IBM Symposium on Mathematical Foundations of Computer Science.* Academic & Scientific Programs, IBM Japan, May, 1980.

[9] Rosenberg, A. L.
Private communication.

[10] Song, S.W.
I/O Complexity and Design of Special-Purpose Hardware for Sorting.
VLSI Document V075, Carnegie-Mellon University, Department of Computer Science, February, 1981.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: CMU-CS-81-111	2. GOVT ACCESSION NO.: AD-A104 739	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle): I/O COMPLEXITY: THE RED-BLUE PEBBLE GAME		5. TYPE OF REPORT & PERIOD COVERED: Interim
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s): HONG, JIA-WEI AND H . T. KUNG		8. CONTRACT OR GRANT NUMBER(s): N00014-76-C-0370
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Carnegie-Mellon University, Computer Science Department, Pittsburgh, PA 15213		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS: Office of Naval Research, Arlington, VA 22217		12. REPORT DATE: February 1981
		13. NUMBER OF PAGES: 19
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)		15. SECURITY CLASS. (of this report): UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

LME